CHAPTER 3

CONTEXT - FREE LANGUAGES

CONTEXT - FREE GRAMMARS

Consider the following regular expression

$$a(a^* \cup b^*)b$$

HOW we can GENERATE IT ?
How it is built?

S - string, M - middle

$a\underbrace{(a^* \cup b^*)}_{M}b$

Rule

$S \to aMb$

string is aMb

How we can now generate M ?

M: $a^* \cup b^*$

$M \to A$
$A \to e$
$A \to aA$

$M \to B$
$B \to e$
$B \to bB$

GRAMMAR - Language GENERATING

Automata - language recognizer
devise

Equiv. def.

$G = (N, \Sigma, R, S)$

REMARK

- $N \cap \Sigma = \emptyset$
- N, Σ alphabets
- Σ - TERMINAL
- N - NON-TERMINAL
- R - set of rules

$R \subseteq N \times (N \cup \Sigma)^*$

In our definition

$V = N \cup \Sigma$ → TERMINALS

$L \subseteq \Sigma^*$

$G = (V, \Sigma, R, S)$

NOTATION:

CAPITAL LETTERS for $N = V - \Sigma$

little letters for Σ.

$\sigma \in \Sigma$, ALWAYS $L \subseteq \Sigma^*$

Write a grammar (CF) generating $L = \{a\}^*$

shorthand $L = a^*$

$G = (V, \Sigma, R, S)$ $L \subseteq \Sigma^*$

$V - \Sigma$ = non terminals $\varepsilon \in \{a\}^*$

$= \{S\}$

$\Sigma = \{a\}$

$R = \{ S \to aS$ (repetition of a), $S \to a$ (way out) $\}$

$\varepsilon \in V^*$ $\notin L(G)$

$L(G) = a^+$

$S, aS, aaS, aaaS \ldots$

$a, aa, aaa, aaaa \ldots$

IS THIS a correct GRAMMAR? NO!

TO GET a^*

$R' = \{ S \to aS, S \to \varepsilon \}$

or add $S \to \varepsilon$ to R.

$G = (V, \Sigma, R, S)$

Why name CONTEXT-FREE?

Rule $r \in R$ $\quad A \to u \quad u \in V^*, A \in V-\Sigma$

says: REPLACE A by u no matter what the strings surrounding A are; replacement is performed INDEPENDENTLY of the CONTEXT of A

Example: Let $x = aaAb \in V^*$ and G has a rule: (r) $A \to aA$

applying the rule (r) to (x) we get $aaaAb$

$y = bbA$ → rule → $bbaA$

DERIVATION IN G

$u \underset{G}{\Rightarrow} v$, $u \Rightarrow v$ (ONE STEP)

(like $(p, u) \vdash_M (q, v)$)

DEF ONE STEP DERIVATION

$u \underset{G}{\Rightarrow} v$ iff there are $A, x, y, v' \in V^*$ such that

① $A \in V-\Sigma$

② $u = xAy \wedge v = xv'y$

③ $A \to v'$ for certain $r \in R$

In plain words ONE STEP DERIVATION

$u \underset{G}{\Rightarrow} v$ iff

we obtain v from u by application of one rule $r \in R$ (in G)

It means

$\exists r \in R \quad \exists x, y, v' \in V^*$

$u = x A y$

$v = x v' y$

and $r = A \to v' \in G$

FORMAL

$$L(G) = \{ w \in \Sigma^* : S \overset{*}{\underset{G}{\Rightarrow}} w \}$$

where $\overset{*}{\underset{G}{\Rightarrow}}$ is a reflexive, transitive closure of G

We write derivations

$S = w_0 \underset{G}{\Rightarrow} w_1 \Rightarrow \dots \underset{G}{\Rightarrow} w_n = w$ or just $w_0 \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_n$

all $w_n \in V^*$

$(w_n = w)$

Example: $G = (V, \Sigma, R, S)$

- $V = \{S, a, b\}$, • $\Sigma = \{a, b\}$
- $R = \{ S \to aSb, \ S \to e \}$

to get $w \in \Sigma^*$

WAY OUT RULE!

Derivations

$S \Rightarrow e \qquad e \in L(G)$

Derivation "way out"

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb \in L(G)$ ($S \to aSb$, $S \to e$)

$S \Rightarrow aSb \Rightarrow ab \in L(G)$ ($S \to e$)

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb \in L(G)$

NOT REGULAR

$$L(G) = \{ a^n b^n : n \geq 0 \}$$

Induction over length of derivation

DEFINITION

L is a CONTEXT-FREE LANGUAGE iff there is a CF grammar G such that

$$L = L(G)$$

REG ⊄ CONTF.

We just proved that

$L = \{a^n b^n : n \geq 0\}$ is a CONTEXT-FREE language that is not regular. GOAL

REGULAR ⊆ C.F

PROPER SUBSET

PROVE LATER Reg ⊆ CF

Example

$G = (V, \Sigma, R, S)$

$V = \{S, a\}$, $\Sigma = \{a\}$

$R = \{S \to aS, \quad S \to e\}$

$e \in L(G)$

$S \Rightarrow aS \Rightarrow aaS \Rightarrow aaaS \Rightarrow aaa \in L(G)$

$L(G) = \{a^* : a \in \Sigma\}$

$L = a^*$ is a Regular language that is context-free

WE ARE GOING TO PROVE

REGULAR LANG ⊆ CONTEXT-FREE

Q1 Is $L = \phi$ a context free language?

Q2. Is $L = \{a\}$ a CF language?

Q3 Is Σ a context free language?

We know that L from Q1-Q3 are regular.

YES!

Answer : construct grammars G_1, G_2, G_3 such that

Q1 $L(G_1) = \phi$

Q2 $L(G_2) = \{a\}$

Q3 $L(G_3) = \Sigma$ - alph.

Observe that Q3 answers Q1 and Q2 for $\Sigma = \phi$, $\Sigma = \{a\}$

Q4 Construct G_4 $L(G_4) = \{e\}$

To get Q1

$L(G_1) = \emptyset$ you must

take $\Sigma \neq \emptyset$ $\boxed{R = \emptyset}$ $G = (V, \Sigma, R, S)$

$S \in V$

In general let $V \neq \emptyset$

$$R \subseteq (V - \Sigma) \times V^*$$

Finite, in particular

$R = \emptyset$.

We can have grammars with $\Sigma \neq \emptyset$, $L(G) = \emptyset$

like in automata!

when $F = \emptyset$

(6d) $V = N \cup \Sigma$ (c

Construct G.

$L(A) = \{a\}$

$G = (N, \Sigma, S, R)$ $S \in N$

$\Sigma = \{a\}$ $N = \{S\}$

$R = \{S \to a\}$

S only one derivation !

$L = \{a\}$

$L(G) = \{a, b\}$

$\Sigma = \{a, b\}$ $N = \{S\}$

$R = \{S \to a, \quad S \to b\}$

no loops !

$L(G) = \{a, b\}$

PROVE: For any Σ, there is CF, G

$L(G) = \Sigma$

$\gamma_0 \to \epsilon$

$\gamma_0 \to \epsilon$

Σ is finite

$\Sigma = \{\sigma_1, \sigma_2 \ldots \sigma_n\}$

Rules

$S \to \sigma_i$ $\quad i = 1 \ldots n$

for $\Sigma \neq \emptyset$

or rule $R = \emptyset$ for $\Sigma = \emptyset$.

When we want $L(G) = \emptyset$

or $S \to \epsilon$ when $\Sigma = \emptyset$

$L(G) = \{\epsilon\}$

Q4 $G_4 = (V, \Sigma, R, S)$

$R \subseteq (V-\Sigma)^* V^*$

Take rule

$S \to e$

Take $V = \{S\}$

take $\Sigma = \phi$

$G = (\{S\}, \phi, S \to e, S)$

$L(G) = \{e\}$

With automata $\Sigma = \phi$
would give $L(M) = \phi$
$\neq \phi$

$\therefore$ $L(M) = e$

$S \Rightarrow a$ or S, a ($a \in L(G)$ iff $S \Rightarrow w$)

$L(G) = \{ w \in \Sigma^* : S \Rightarrow w \}$

or $S \Rightarrow A$

$S, A, aAbb$ ($\in V^*$, not Σ^*) $\Rightarrow aabbbb$ derive in G

$S, A, aAbb$ remains $aAbb$, $aabbbb$

initial symbol

$L(G) = \{ a^n b^{2n} : n \geq 1 \} \cup \{a\}$

$G = (\{S, A, a, b\}, \{a, b\}, R, S)$

V Σ

R is a set of 2 rules

r_1 $S \rightarrow a$

r_2 $S \rightarrow A$, $A \rightarrow aAbb$ (LOOP)

$A \rightarrow e$

EXAMPLE 2

$$G = (V, \Sigma, R, S)$$

$V = \{a, b, S\}$ → non terminal

$\Sigma = \{a, b\}$ $\quad V - \Sigma = \{S\}$

$$R = \{S \to aSb, \quad S \to a\}$$

↓ define Σ, V of G

$$R \subseteq (V - \Sigma)^* V^*$$

$$L(G) = \{a^{n+1} b^n : n \geq 0\}$$

$a \in L(G)$ $\quad n = 0$

$a \in L$

$aab \in L$

$aaabb \in L$

$aaaabbbb \in L$

Pattern !

Ex 2

guess !

$$L(G) = \{ a^{n+1} b^n : n \geq 0 \}$$

Exercise

Prove by math induction over length of derivation that

$\forall n \in \mathbb{N} \quad a^{n+1} b^n \in L(G)$

EXAMPLE 3

$G = (V, \Sigma, R, S)$

Ex 3 eng Gh

$V = \{S, (,)\}$

$\Sigma = \{(,)\}$

$R = S \to e \mid SS \mid (S)$

three rules

$S \to e$ $\quad$ $S \to (S)$

$S \to SS$

TASK: Prove $L(G)$ is not Regular

Example 3

PROVE: L(G) NOT REGULAR

$G = (V, \Sigma, R, S)$

$V = \{S, (,)\}$, $\Sigma = \{(,)\}$

R: ① $S \to e$, ② $S \to SS$, ③ $S \to (S)$

Some derivations

① $S \overset{2}{\Rightarrow} SS \overset{2}{\Rightarrow} SSS \overset{3}{\Rightarrow} (S)SS \overset{3}{\Rightarrow} ((S))SS$

$\overset{3}{\Rightarrow} ((S))(S)S \overset{3}{\Rightarrow} ((S))(S)(S) \overset{1}{\Rightarrow} ((\,))(S)(S)$

$\overset{1}{\Rightarrow} ((\,))(\,)(S) \overset{1}{\Rightarrow} ((\,))(\,)(\,) \in L(G)$

② $S \to (S) \Rightarrow ((S)) \Rightarrow (((S))) \to ((()))\in L(G)$

produces $(^m)^m$

In particular

$L_1 = \{(^n)^n : n \geq 0\} \subseteq L(G)$ (n - length of clan).

L_1 is not regular $\to$ (PROVE) L(G) NOT REGULAR

CONTRADICTION PROOF: Assume L(G) regular

$L_1 = L(G) \cap (^*)^*$ $\quad L = (^*)^*$ is REGULAR

Regular lang. closed under $\cap$, so

L_1 REGULAR, CONTRADICTION.

L(G) not regular.

Show that

$$L = \{ a^m b^n : m > n \}$$

is CONTEXT FREE.

This means:

CONSTRUCT a CF grammar G

such that

$$L(G) = L$$

G must "produce" always more a's than b's.

Example 4

Show that a language

$$L = \{a^m b^n : m > n\}$$

is CONTEXT-FREE. $L_1 = \{a^m b^n : m \geq n\}$ is also CF

We will define G, such that

$L = L(G)$

$G = (V, \Sigma, S, R)$

$\Sigma = \{a, b\}$ $N = \{S, A, B\}$ (non-terminals)

$V = \Sigma \cup N$

$R = \{ S \to AB$

$A \to a \mid aA$ (Two rules $A \to a$, $A \to aA$)

$B \to aBb \mid e \}$ $B \to e$

IDEA: we want to generate more a's than b's and b's must follow a's.

PUT: A – for creating a, B – for creating a, b

- $S \to AB$ (start creating a's and b's)
- $A \to aA$ create a's
- $A \to a$ make string of a's $\in L$ (more a's)
- $B \to aBb$ CREATE b's AFTER a's
- $B \to e$ Make $w \in L$ ("WAY out")

Why we can't have

$A \to e$ as a rule?

$S \Rightarrow AB \Rightarrow B \Rightarrow aBb \Rightarrow ab \in L(G)$

by rule $A \to e$

by $B \to aBb$

$B \to e$

so we get $ab \in L(G)$

what's the case

$n = n - 1$

not $n \neq n$

$R = \{ S \xrightarrow{1} AB,\ A \xrightarrow{2} aA \mid \xrightarrow{3} a,\ B \xrightarrow{4} aBb \mid \xrightarrow{5} e \}$

Some derivations

Rules			
S		S	S
AB	1	AB	AB
aAB	2	aB	aB
aaB	3	aaBb	a
aaaBb	4	aab	BASE
aaab	5	n=2, n=1	n=1, n=0
n=3, n=1			M ? M Add

$A \to e$ get $m = n$

$$L(G) = \{ a^m b^n : m > n \}$$

CONTEXT-FREE GRAMMARS ARE CREATION OF N. CHOMSKY (1956)

Proof by induction over # STEPS in the derivation.

EXAMPLE

$V = \Sigma \cup N \qquad \Sigma \cap N = \phi$

$G = (\Sigma, N, R, S)$

Σ - TERMINALS
N - non-terminals

$\Sigma = \{$ Jim, big, green, cheese, ate $\}$

$N = \{ S, A, N, V, P \}$

S - SENTENCE
A - ADJECTIVE
N - NOUN
V - VERB
P - PHRASE

This is how GRAMMARS started:
Rules of NATURAL Language
CHOMSKY 1956

$\Sigma = \{$ Jim, big, green, cheese, ate $\}$

$W = \{ S, A, N, V, P \}$

$R = \{ S \to PVP,\ P \to N,\ P \to AN \}\ \cup$

{ Sentence → Phrase Verb Phrase
Phrase → Noun, Phrase → Adjective Noun }

$\cup \{ A \to$ big, $A \to$ green, $N \to$ cheese
$N \to$ Jim, $V \to$ ate $\}$

{ Adjectives: big, green
Nouns: cheese, Jim
Verb: ate }

Some Derivations:

- Jim ate cheese
- big Jim ate green cheese
- big cheese ate Jim

But G describes "too much"

for example

- big cheese ate green green big green big cheese
- green Jim ate green big Jim etc

OBSERVE

Rules in a CF Grammar

are:

$A \to u^*$ $\quad u \in V^*$

When you search for derivations, you always get SIMPLER expressions

Search: use INVERSE rules

$u^* \to A$

Replace u^* (in w) by

A (u w) (always)

search is not unique so

but WE'LL FINISH until final (S)

START

Results from late 70ties

① Class of procedure free languages is Regular

② Class of prog. lang with procedures is CF

~~Automata proof~~

Prop Calc = reg.

Pred Calc = CF.

COMPUTER PROGRAMS (in any programming language) must satisfy some RIGID criteria in order to be SYNTACTICALLY CORRECT

SYNTAX of most programming languages can be captured by CONTEXT-FREE Grammars

PARSING a program - way to analyze it in order to understand its SYNTAX

CF grammars are used for PARSING (p122-129**)

Example (p117)

→ a grammar that generates a FRAGMENT of many progr. languages.

$L \subseteq \{(,), +, *, id\}^*$

id stands for IDENTIFIER (variable name)

Discovering identifiers (or reserved words of the language, or numerical constants) is accomplished at the earlier stage of LEXICAL ANALYSIS by algorithms BASED ON REGULAR EXPRESSIONS and FINITE AUTOMATA

$G = (V, \Sigma, R, E)$

E - START expression
T - terms
F - factor

$V = \{+, \cdot, (,), id, T, F, E\}$

$\Sigma = \{+, \cdot, (,), id\}$

$R = \{ E \xrightarrow{1} E+T,\ E \xrightarrow{2} T,\ T \xrightarrow{3} T \cdot F,\ T \xrightarrow{4} F,\ F \xrightarrow{5} (E),\ F \xrightarrow{6} id \}$

Derivation of a string $\in L(G)$

$(id \cdot id + id) \cdot (id + id)$ SEARCH

Which rule applied in INVERSE ↕ 6 ↑ DERIVATION look BACK for derivation

CF GRAMMARS are good for SEARCH of derivations

$(F \cdot id + id) \cdot (id + id)$

Search from MOST LEFT SIDE

↕ 6

$(F \cdot F + id) \cdot (id + id)$ INVERSE $F \to T$ rule $T \to F$

↕ 4

$(T \cdot F + id) \cdot (id + id)$ INVERSE $TF \to T$

↕ 3 Rule $T \to TF$

$(T + id) \cdot (id + id)$ $T \to E$

↕ 2 Rule: $E \to T$

$(E + id) \cdot (id + id)$

E-START → GOAL → Get E

$R = \{ E \xrightarrow{1} E+T,\ E \xrightarrow{2} T,\ T \xrightarrow{3} T \cdot F,\ T \xrightarrow{4} F,\ F \xrightarrow{5} (E),\ F \xrightarrow{6} id \}$

$(E + id) \cdot (id + id)$

$\updownarrow 6$

$(E + F) \cdot (id + id)$

$\updownarrow 4$

$(E + T) \cdot (id + id)$

$\updownarrow 1$

$(E)\ (id + id)$

$\updownarrow 5$

$F\ (id + id)$

$\downarrow 4$

$T\ (id + id)$

$\downarrow 6$

$T\ (id + F)$

$\downarrow 4$ need T

$T\ (id + T)$

$\downarrow 6$

$T\ (F + T)$

$\downarrow 4$

$T\ (T + T)$

$\downarrow 2$

$F\ (E + T)$

GOT IT!

follow

Rule 6: $F \to id$ — Inverse: $id \to F$

Heuristic = simplify

$\downarrow$ E START

STRATEGY

Need to transform it to $T\ (E + T)$ then

$T\ (E + T)$

$\downarrow 1$

$T\ (E)$

$\downarrow 5$

$T\ F$

$\downarrow 4$

T

$\downarrow 2$

E START

END of search!